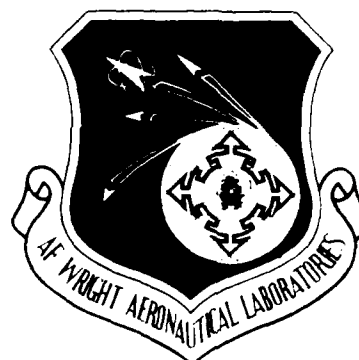


AD-A168 764

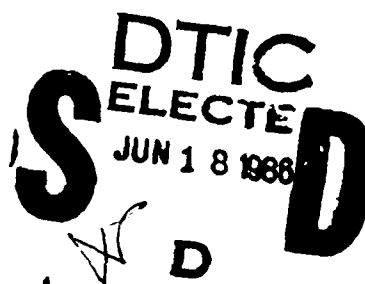
2

AFWAL-TR-84-1186



A NETWORK ARCHITECTURE FOR DATA-DRIVEN SYSTEMS

James E. McDonald
System Evaluation Branch
Systems Avionics Division



July 1985

Final Report for Period 1 August 1982 - 1 August 1984

Approved for public release; distribution unlimited.

OTIC FILE COPY

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6553


3 7 1984

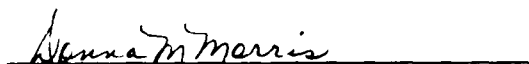
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

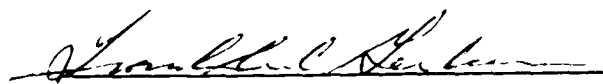
This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.


JAMES E. McDONALD
Project Engineer
Logistics Technology Group


DONNA M. MORRIS, Actg Chief
System Evaluation Branch

FOR THE COMMANDER


FRANKLIN C. GERKEN, LT COL, USAF
Deputy Chief
System Avionics Division
Avionics Laboratory

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFMHL/AAAF, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS NONE		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT PUBLIC RELEASE; Distribution Unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-84-1186		5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Avionics Laboratory	6b. OFFICE SYMBOL (If applicable) AFWAL/AAAF	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State and ZIP Code) AFWAL/AAAF Wright-Patterson AFB OH 45433		7b. ADDRESS (City, State and ZIP Code) AFWAL/AAA Wright-Patterson AFB OH 45433		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State and ZIP Code)		10. SOURCE OF FUNDING NOS.		
		PROGRAM ELEMENT NO. 62204F	PROJECT NO. 2003	TASK NO. 02
11. TITLE (Include Security Classification) Network Architecture for Data-Driven Systems		WORK UNIT NO. 69		
12. PERSONAL AUTHOR(S) McDonald, James E.				
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM 1 Aug 82 TO 1 Aug 84	14. DATE OF REPORT (Yr., Mo., Day) July 1985	15. PAGE COUNT 72	
16. SUPPLEMENTARY NOTATION The software referenced herein does not reflect Air Force-Owned or developed computer software.				
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB. GR.		
09	02			
		Data Driven Distributed Systems Computer Networks Real Time		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This technical report discusses the hardware and software constructs of a unique computer network designed by the author. The network is especially designed for a multiple computer system operating in a synchronous real-time simulation system environment. The network emulates a shared memory system but has the unique property of being serially linked. All currently available shared memory systems utilize parallel address and data transmission schemes that demand a close proximity of the computers. Now, due to the serial linking, the computers may be hundreds or thousands of meters apart. Innovative data-driven hardware and software concepts applied to the serially-linked shared-memory multi-computer structure allow network transmission and subsequent software execution to be commanded relative to the dynamics of shared data variables. This concept is contrasted to synchronous network transmission and software execution of conventional real-time multi-computer systems.				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL James E. McDonald		22b. TELEPHONE NUMBER (Include Area Code) 113 255 6543	22c. OFFICE SYMBOL AFWAL/AAAF	

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED

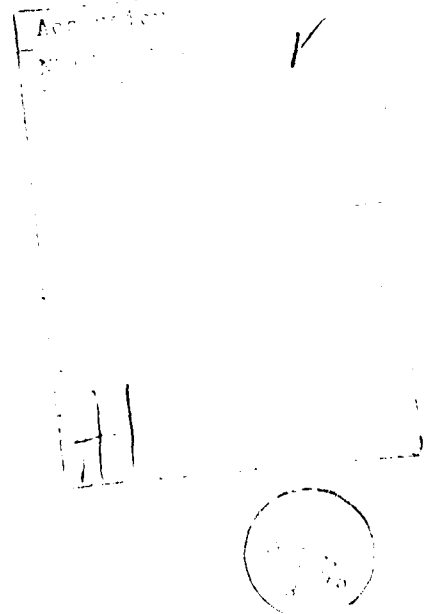
SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

	Page
List of Figures	vi
List of Abbreviations	vii
Preface	viii
 Section	
1. INTRODUCTION AND BACKGROUND	1
Computer Networks	1
Prevalent Data-Driven Architectures	8
Real-Time Systems	9
A Multi-Computer Real-Time Flight Simulator	10
2. DATA-DRIVEN SYSTEM REQUIREMENTS AND DESIGN SKETCH	13
Serial Communication Scheme	17
Serial/Parallel Transmission and Reception	17
Distributed Shared Memory	17
Information Detection	17
Data Address Vectored Interrupt Scheme	18
3. NETWORK HARDWARE CONCEPT ELABORATION	19
Serial Communication Scheme	19
Local Shared Memory Serial Interface Unit	23
4. DATA-DRIVEN SOFTWARE CONCEPT ELABORATION	26
Real-Time Operating System	26
Secondary Memory Utilization	26
Data Flow Graphical Programming	27

TABLE OF CONTENTS (CONTINUED)

	Page
High Order Language Implementation	27
Data-Driven Real-Time Flight Simulator Software	28
5. ADVANCED SYSTEM ELABORATION	31
Single Computer System	31
Data Variable Sensitivity	32
Real-Time Graphics Implementation	34
Multi-Variable Synchronization	34
Data Tagging	35
Serial Transmission Error Detection and Correction	35
6. PROTOTYPE DESIGN	37
Hardware Structure	39
Hardware Operations	40
7. SUMMARY AND CONCLUSIONS	43
8. APPENDIX I - PROTOTYPE HARDWARE SCHEMATICS	45
9. APPENDIX II - PROTOTYPE MICROCONTROLLER FIRMWARE	52
10. BIBLIOGRAPHY	61



LIST OF FIGURES

Figure	Page
1.1 Shared Memory Network Configuration	2
1.2 Hierarchy Network Configuration	4
1.3 Bus Network Configuration	4
1.4 Star Network Configuration	6
1.5 Ring Network Configuration	7
1.6 Multi-Computer Flight Simulator	11
2.1 Serially-Linked Shared-Memory Ring Network Configuration .	15
2.2 Local Shared Memory Serial Interface Unit Block Diagram . .	16
3.1 Serial Information Word Format	19
3.2 Ring Network Physical Layout	21
3.3 Serial Message Transmission Time	22
4.1 Flight Simulator Data Flow Graph	29
5.1 Single Node Data-Driven Computer System	31
5.2 LSMSIU with Data Sensitivity Parameter Implementation . . .	33
6.1 M68000 Based Data-Driven Network Node Prototype Design Block Diagram	38

LIST OF ABBREVIATIONS

ALU	Arithmetic Logic Unit
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access with Collision Detecion
DAVI	Data Address Vectored Interrupt
DTACK	Data Transfer Acknowledge
ECB	Educational Circuit Board
EPROM	Eraseable Programmable Read Only Memory
FDM	Frequency Division Multiplexing
FIFO	First In First Out
INFO	Information
IRQ	Interrupt Request
I/O	Input/Output
LSM	Local Shared Memory
LSMSIU	Local Shared Memory Serial Interface Unit
MEM	Memory
MSB	Most Significant Bit
NIU	Network Interface Unit
OTW	Out The Window
RAM	Random Access Memory
ROM	Read Only Memory
SLSM	Serially Linked Shared Memory
TDM	Time Division Multiplexing

PREFACE

The initial goal of this project was to fully design a multiple computer shared memory system with a serial, rather than the conventionally parallel, transmission link. The shared memory was to be distributed with each computer communicating with its own copy of shared memory. Reads from the local copy of shared memory would occur immediately while writes to shared memory would require a serial transmission on the network to update all distributed shared memory copies.

In the associated research and development process, data-driven architectural possibilities were discovered and applied to the serially-linked shared-memory system under development. In the final analysis, the data-driven concept application superseded the importance of initial concepts and was by far the most exciting aspect of the project.

The project started mid 1982 and has experienced intermittent progress. In late 1983, it was discovered that similar research had been undertaken elsewhere [4], [5]. These research goals did not include data-driven structures and were simply defined to design a high speed multiprocessor system.

This report has been difficult to formulate to assure a clear understanding by the reader. The problem results from the multi-level structure of the system being explained. The top level system must be

understood before subsystem interactions become clear and the subsystems must be understood before the top level system is comprehended. This anomaly forces the author to repeat the concept explanations in many cases.

The proposed system requires the user-programmer to understand inherent limitations. In shared memory systems, errors occur when multiple computers write to the same location simultaneously. The writes are actually performed sequentially and the program in each computer can not detect the sequence order. However, one write is last and it remains as the memory data until overwritten. The same error occurs in replicated shared memory systems. The error is manifested by the fact that each copy of shared memory may have a different value for the simultaneous write case. It is expected that this fact would create havoc in the executing program. Another concern is the scheduling of data-driven software in multiple computer systems. A program is to be executed when data inputs become available, i.e. change value. Inputs to a program may come from multiple unsynchronized computers at unpredictable times. Without proper synchronization control, transient errors in final and intermediate data outputs may occur. Multi-computer synchronization is therefore mandatory.

1. INTRODUCTION AND BACKGROUND

The objective of this project is to develop hardware and software concepts supporting a data-driven serially-linked shared-memory network for real-time applications. A small prototype demonstrator is to be designed to support the concept development.

This project encompasses the digital system, computer network and real-time application subjects. During the course of this document, the author will introduce and support ideas that blend the three mentioned subjects into a single innovative architecture. The 'glue' for this blend is a unique data-driven hardware and software processing scheme. To set the stage, the applicable research performed in support of this project is now summarized.

COMPUTER NETWORKS

An important subject being actively researched in computer science today is computer networking. Networks with performance measured in terms including speed, cost, utility, efficiency, expandability and reliability are currently being developed.

Networks are transmission media that permit communication between two or more devices. The transmission media can be implemented by electrical and fiber-optic means and are categorized into many types including (1) Shared Memory, (2) Hierarchy, (3) Bus, (4) Star, and (5) Ring. As these network topologies are examined, the final parameters of the network being designed must be developed. The initial

attributes of the desired network are:

(1) Shared random-access memory emulation, (2) Serial fiber-optic transmission medium, and (3) Near autonomous operation of each networked computer. Attributes 1 and 2 conventionally imply a serial message scheme with a word count of two words per message; one word indicating the data address and the second word being the data.

The conventional shared memory network configuration is illustrated in figure 1.1. Shared memory networks support parallel communication among the networked computers by allowing all computers to access a single common centralized memory. Any computer can access any word of the shared memory at any time. Specified shared data words can be arranged as semaphores to facilitate communication among the computers. In some shared memory units, circuitry exists that enables one computer to interrupt another computer for synchronization purposes.

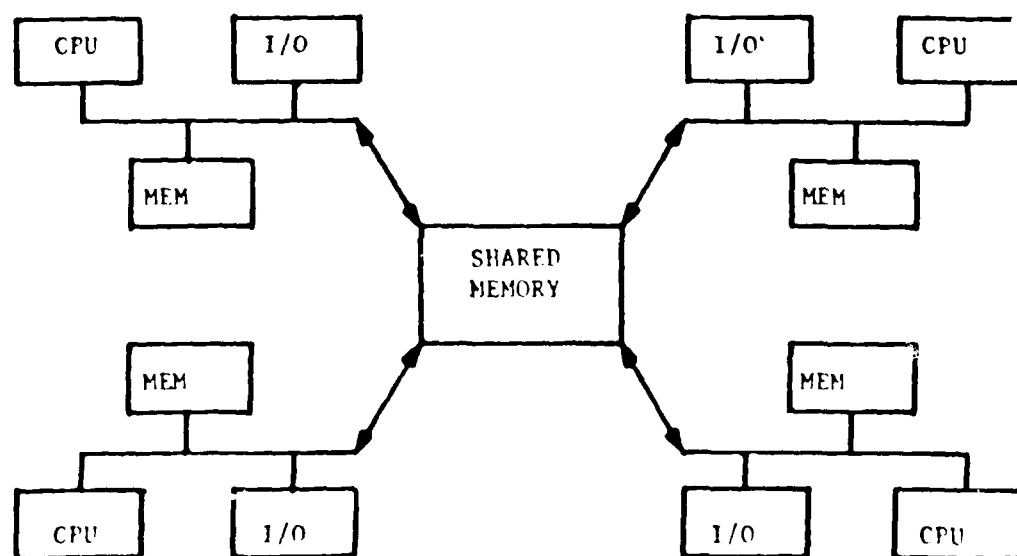


Figure 1.1 Shared Memory Network Configuration

Shared memory transmission media consist of typically 16 to 24 unidirectional parallel address lines, 16 to 32 parallel bidirectional data lines and 3 to 10 control lines. Transmission speed is typically 1 to 10 microseconds per single word transfer. Physical spacing between the computers is limited to tens of feet due to the high speed parallel transmission requirements, and electrical transmission line characteristic limitations. There is no theoretical limit to the number of computers able to be linked in a shared memory system. However, shared memory access time increases with the number of actively involved computers. Therefore shared memory systems typically connect sixteen or less computers.

Real-time software utilizations of shared memory typically configure the 'common block' of shared data variables in the shared memory area. This configuration permits easy distribution of data among the various software modules.

Hierarchy network configurations are illustrated in figure 1.2. Hierarchy networks can be implemented by shared memory networks where all networked computers will not access a single centralized shared memory. There may be layers of shared memory networks to be pierced before one computer may 'talk' to another computer. This structure was demonstrated in the Cm* project [3] at Carnegie Mellon University where 50 PDP-11 computers were linked in a hierarchial network structure.

A bus network configuration is illustrated in figure 1.3. Bus networks support digital communication by one of the following schemes:

- (1) time division multiplexing (TDM),
- (2) frequency division multiplexing (FDM),

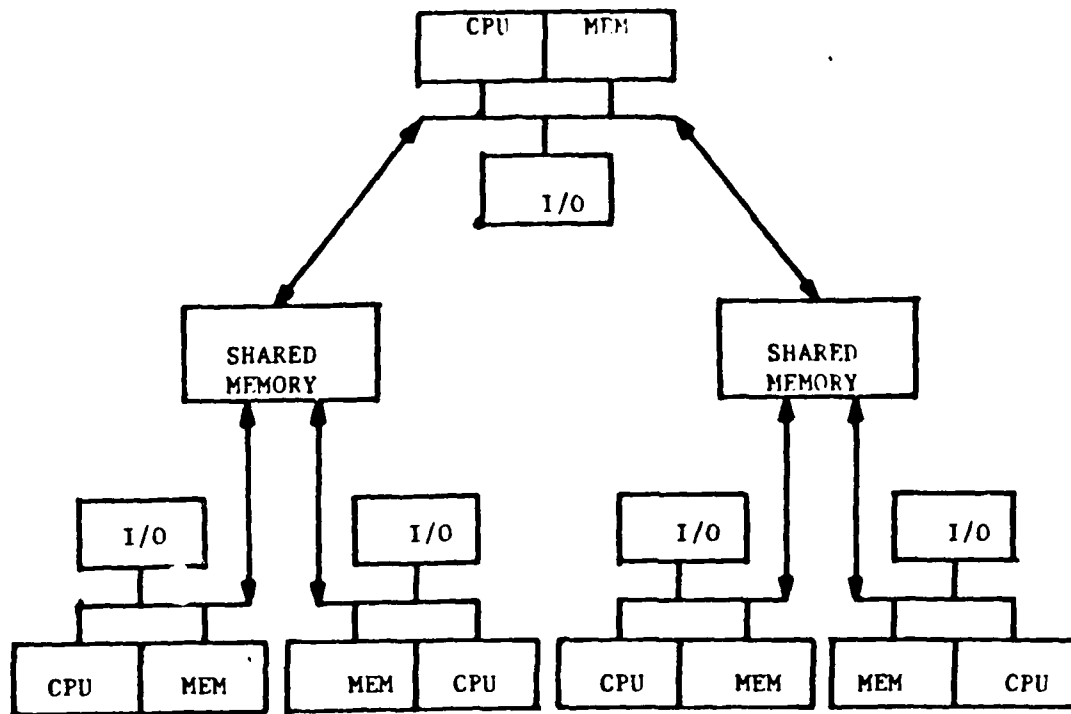


Figure 1.2 Hierarchy Network Configuration

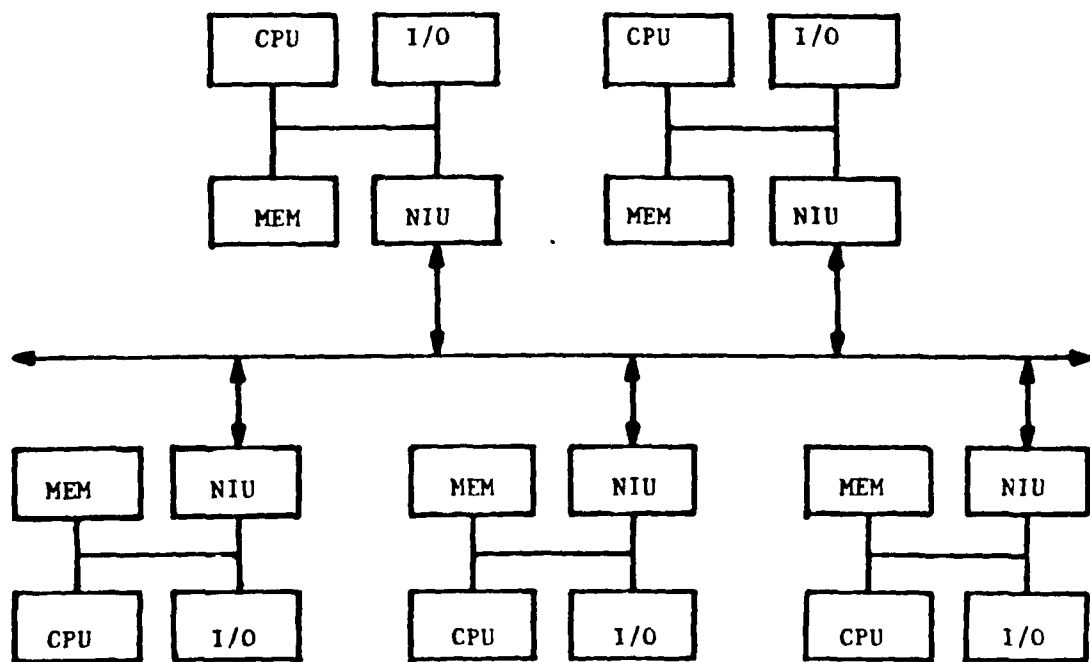


Figure 1.3 Bus Network Configuration. (NIU = Network Interface Unit)

(3) carrier sense multiple access with collision detection (CSMA/CD).

(4) token passing bus master control.

The TDM scheme controls bus communication by allowing a single node to transmit data during a specified repetitive time slot.

The FDM scheme controls bus communication by assigning a specified carrier frequency to each transmitting and receiving node pair. This is similar to the control of the radio frequency airwaves.

The CSMA/CD is a prevalent way to implement local area networking. The scheme allows any node to transmit its message when it detects no traffic on the bus. If by chance, two or more nodes transmit nearly simultaneously a collision occurs and they immediately quit and perform retransmissions at different delay times later. This system is efficient when transmitting large data messages but not for small data messages.

Token passing controls bus communications by allowing only the node with the token to transmit. When communication is complete, the token is passed to a node that needs to transmit. Token passing for few word messages is not efficient. Since the message packet of the thesis project is anticipated to be only two words long, CSMA/CD and token passing were dropped from further consideration.

A star network configuration is illustrated in figure 1.4. Star networks also permit communication by TDM or FDM structures. All nodes receive information simultaneously. A transmitting node must know when to transmit or on what frequency band to transmit. This star structure is ideal for fiber-optic systems with limited receiving dynamic range which is currently a significant problem for fiber-optic bus

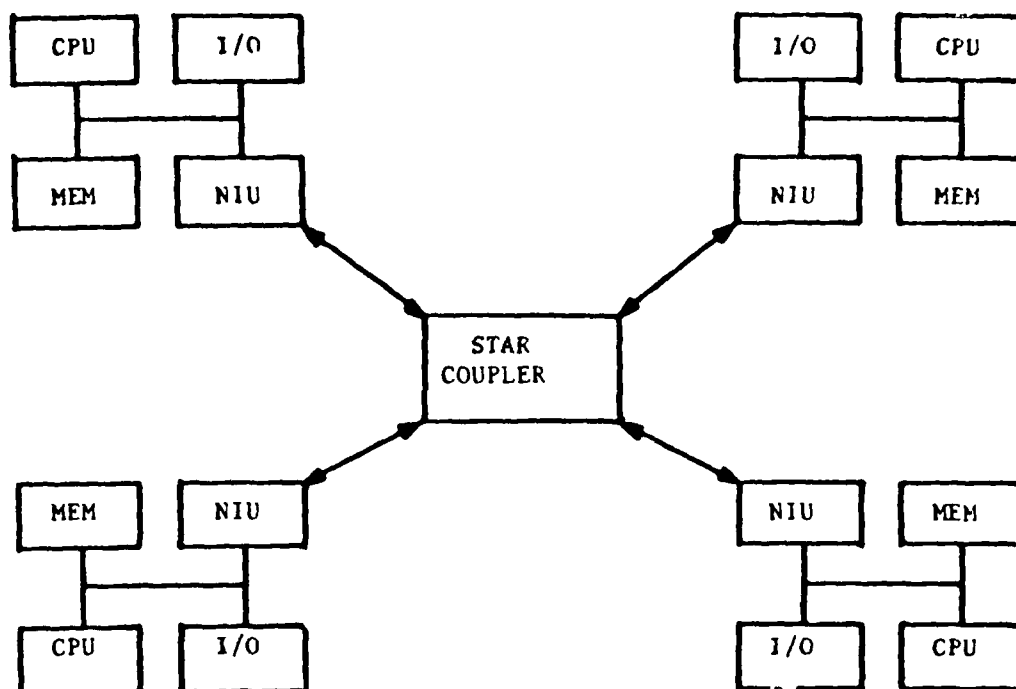


Figure 1.4 Star Network Configuration

configurations. The central star connector is a passive device. However by inserting 'smart' multiplexing circuitry, the star could govern all network transmissions allowing near-autonomous operation at each star network tip. Therefore TDM or FDM protocols would not be required.

A ring network configuration is illustrated in figure 1.5. Ring networks permit data transmission around a ring of networked computers. Each node on the ring stores incoming data and then forwards the data plus any self generated data to the next node on the ring. The node that generates a data message must know not to forward that particular message when received. A source node number field in the transmission packet is therefore employed for this purpose. All nodes receiving message examine the source field to determine the source number and

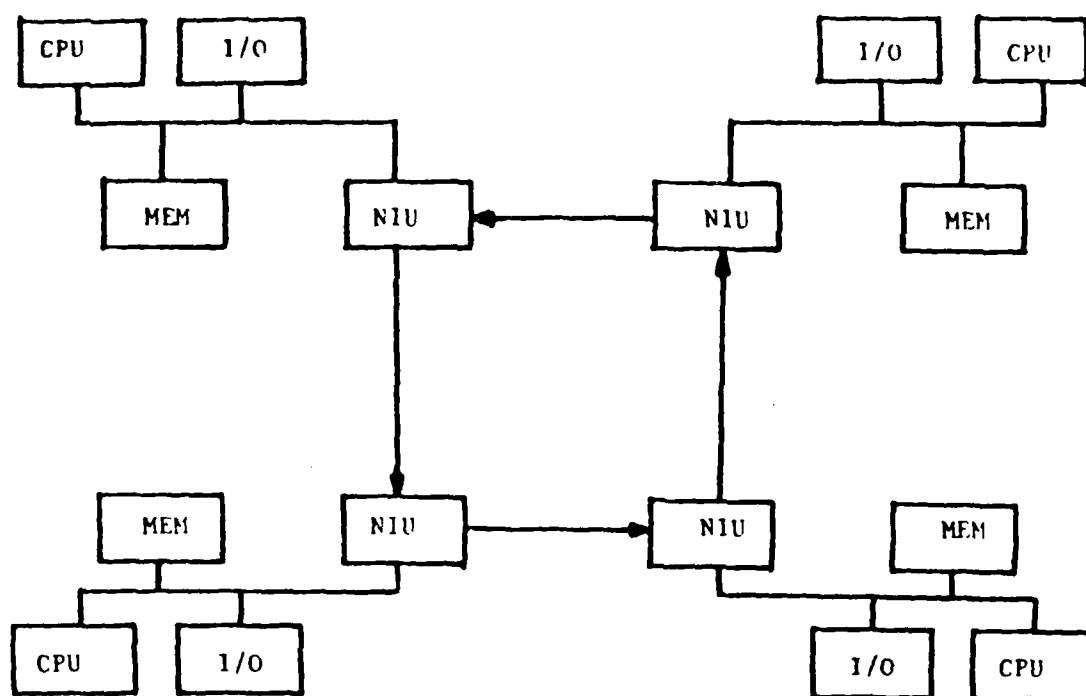


Figure 1.5 Ring Network Configuration

decide whether or not to forward the message.

Token passing is an additional method to determine which node may talk on the ring network. Similar to a source number field is a token field. When a node receives a message with the token field empty (e.g. all zeroes) he may not talk or append messages on the ring. Conversely, if the node desires to talk, it must discover a message packet with a full token field, forward that message with an empty token field and then append any messages to the ring. The last message the node transmits must have a full token field. Ring protocols determine the maximum transmission time or volume a node may transmit before relinquishing the token. If no node has data to transmit, a null message with a full token field must be constantly transmitted around the ring.

Soon to be blended into the initial ring network configuration is the author's version of a data-driven network architecture. However, the prevalent data-driven, Data Flow architecture should be first discussed in order not to confuse the subjects.

PREVALENT DATA-DRIVEN ARCHITECTURES

Currently available data-driven architectures are not similar to the data-driven architecture proposed by this author. These architectures are in line with Data Flow technology whereby a software program is executed when the data inputs to the program become available. The hardware behind this technology is structured as a large number of very tightly coupled arithmetic logic units (ALU's). An ALU is presented a program when data inputs to the program are available. This large numbered parallel ALU structure is a product of the following philosophy: Since integrated circuit manufacturing technology is very close to the theoretical limits, the only way to improve the performance of a computer is to compose the computer with a large number of separate processing elements. Data Flow software technology is being developed to efficiently use the numerous processing elements as a very high speed computer. The architecture is constructed to take advantage of parallel structures in the software. That is, all parallel elements of the software structure are executed simultaneously. Repeating, the proposed data-driven architecture is not related to the data-driven Data Flow technology, although there is no reason the two systems could not be integrated. The author's version of 'data-driven' simply stated is that a program is scheduled for execution when any of the program's data inputs change value.

REAL-TIME SYSTEMS

The connotation of 'real-time' in computer science terminology varies from user to user. Simply, a real-time computer program must execute at a rate greater than or equal to its input data rate. While a computation response time of five seconds may be considered real-time for one user group, one millisecond may be considered an inadequate response time for another user group. Analog computers are inherently real-time by reason that their summing and multiplying functions are considered instantaneous and that analog integration is solely deterministic time dependent. Von Neuman computers are not inherently real-time and neither are the associated computer networks. A real-time digital computer product is a function of the applied software rather than the hardware. The hardware only performs its programmed tasks as fast as technologically possible. The real-time digital computer program simply repeats a calculation sequence that approximates an analog process. The repetition rate is chosen so that the calculation outputs appear continuous to the observer.

Shared memory systems for real-time application of multiple computer systems permit quick and random access to shared data. The data in the shared memory is typically structured as a Fortran common block. The shared common block may contain various data arrays and assorted variables. These arrays and variables never change their physical address during the run of a real-time program and are never swapped to/from secondary storage as may be the case in a single or multiple central processing unit (CPU) interactive but non-real-time application. The multiple computer shared common block provides an easy way to declare data to be shared between several CPU's.

A MULTI-COMPUTER REAL-TIME FLIGHT SIMULATOR

The real-time system of concern can be best explained as a multi-computer flight simulator. The flight simulator structure is shown in figure 1.6. The pilot sitting in the cockpit views a dynamic out-the-window (OTW) scene that simulates the actual OTW scene during flight. One computer interfaces the cockpit joystick, throttle and I/O devices, another computer generates the graphics while the main computer calculates the equation of flight motion. The entire scheme operates at a synchronous repetition speed so as to appear to be an analog function to the pilot. This speed is normally 30 to 100 repetitions per second.

Considering the software to be basically configured as a synchronous and repetitive polling structure and that apriori pilot commanded inputs are not known and that fast response time is required, the entire program must remain in each computer's main memory. There is insufficient time for swapping programs to/from secondary memory no matter how infrequently used.

To summarize the shortcomings of the flight simulator computer configuration is to list the problems that the proposed data-driven information network will solve. Namely,

- (1) Due to the shared memory system, the computers must be physically close.
- (2) Due to the real-time software structure, large main memories are required and secondary memories are not utilized.
- (3) Computational power is continuously expended regardless of varying workload requirements of the real-time system. For example, OTW scenery is more dynamic when flying fast and low than flying high and

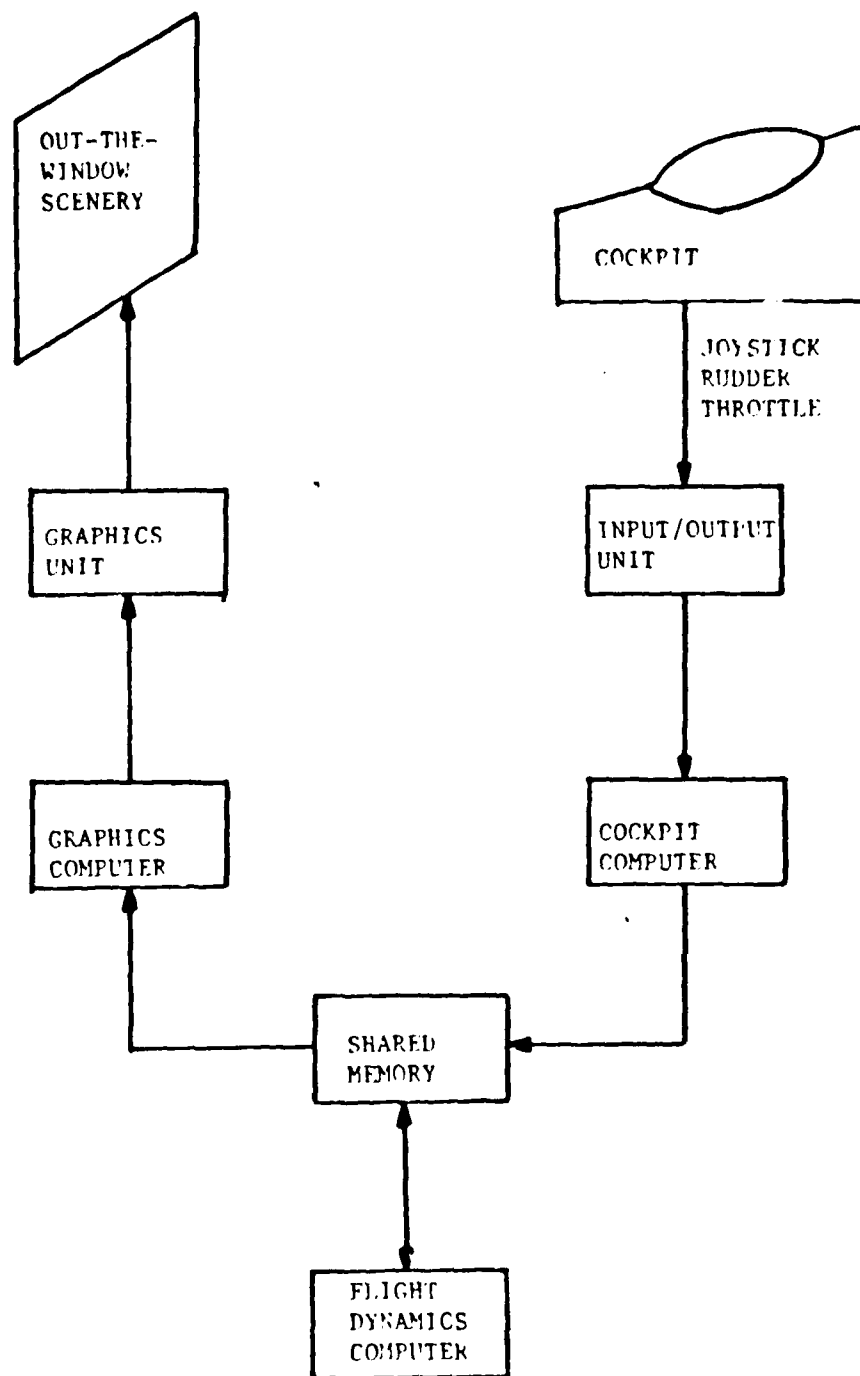


Figure 1.6 Multi-Computer Flight Simulator

slow. Workload requirements are a function of the OTW scenery dynamics.

(4) Shared memory network bandwidth is continuously consumed, again regardless of varying workload requirements.

2. DATA-DRIVEN SYSTEM REQUIREMENTS AND DESIGN SKETCH

After observing current network capabilities and program operation deficiencies in the real-time arena, the requirements and design of the data-driven information network for real-time multiple computer systems was formalized.

The network requirements are derived from an optimized blend of real-time application of available network components. The requirements are:

(1) Shared memory emulation. To the host computer operating software, the network must appear as a shared memory unit which implies that the network is nearly transparent. This will permit an autonomous or near autonomous program operation. Reads and writes to the shared memory shall require no more host CPU time than conventional shared memories which is approximately the same time as main memories (.5 to 1 microsecond). To increase the thruput of the shared-memory network, a copy of the shared-memory is to be located at each networked computer. Shared memory reads would involve only the computers locally owned copy of the distributed shared memory while shared memory writes would update all copies of distributed shared memory.

(2) Serial linking. To avoid bulky and limited distance shared memory cables, high speed fiber-optic serial links shall be employed. Shared memory transfer times shall be accommodated and with fiber-optics, the distance between computers can be easily up to 10 kilometers.

(3) Information only transfer. To increase the efficiency of the network, the only items transmitted on the network are data variables that have been updated, or written to with new values. These new values are termed 'information'. A data variable can only acquire a new value as result of a write action by a CPU, or direct memory access (DMA), to the particular data variable. Therefore only write actions instigating new data variable values can instigate a network transmission. The information only transfer gives birth to the data-driven concepts of the entire system architecture. It should be remembered that in conventional real-time systems of concern, the majority of write actions to shared data variables do not modify the variable value but would still demand a network transmission.

(4) Data address vectored interrupt (DAVI). When a computer receives information from the network, an interrupt shall be generated by the receiving node hardware directing the receiving computer to the software needed to process the information. This technique is employed to increase efficiency and to minimize the response time of the reacting software program.

With these requirements in mind, the initial block diagram design of the Serially-Linked Shared-Memory (SLSM) ring network is illustrated in figure 2.1. A detailed block diagram of the network node is shown in figure 2.2. These block diagrams illustrate the major component configurations: (1) Serial Communication Scheme, (2) Serial/Parallel Transmission and Reception, (3) Distributed Shared Memory, (4) Information Detection, and, (5) Data Address Vectored Interrupts. The grouping of each local copy of the Distributed Shared Memory with Serial/Parallel Transmission and Reception, Information Detection, and

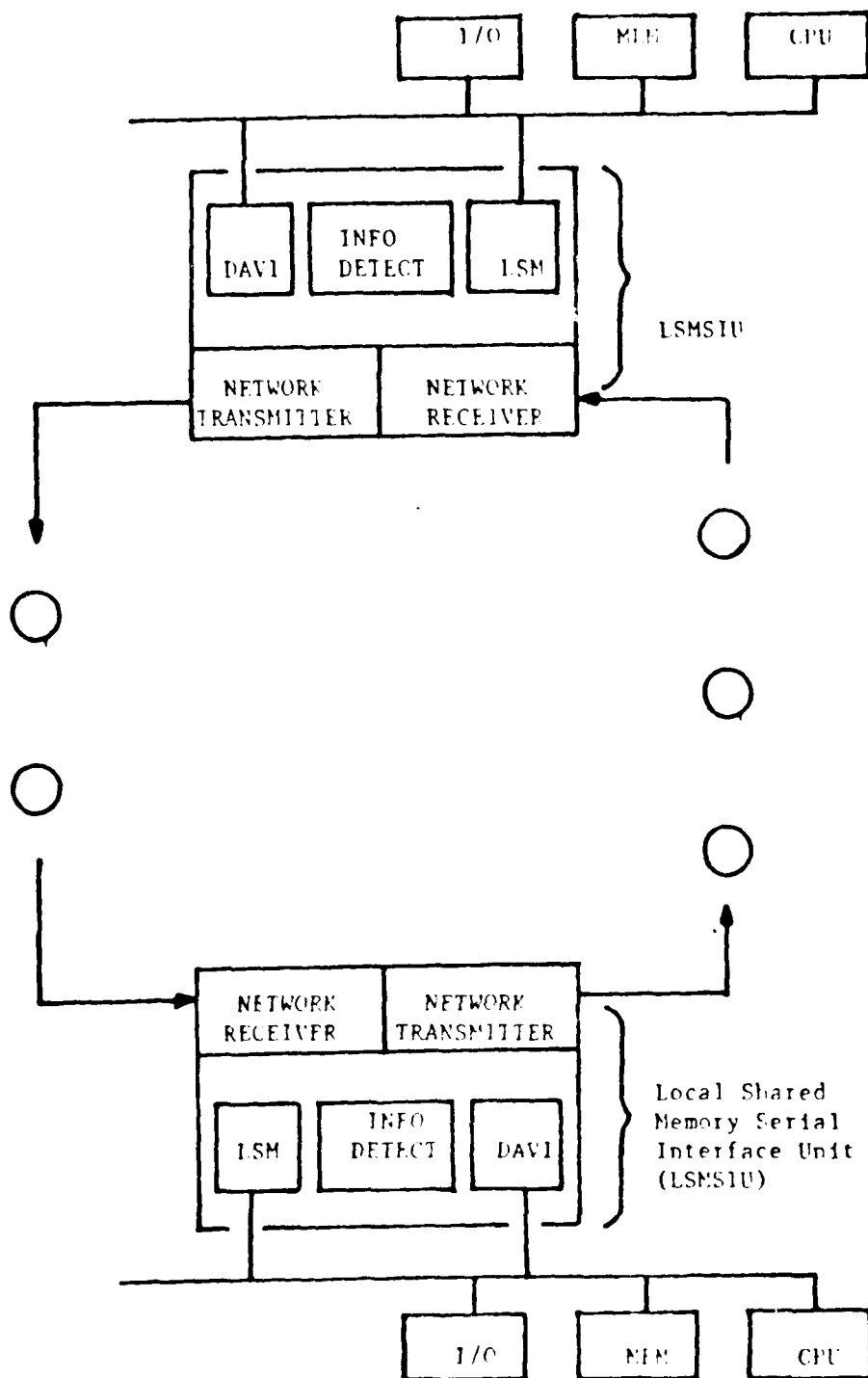


Figure 2.1 Serially Linked Shared Memory Ring Network Configuration

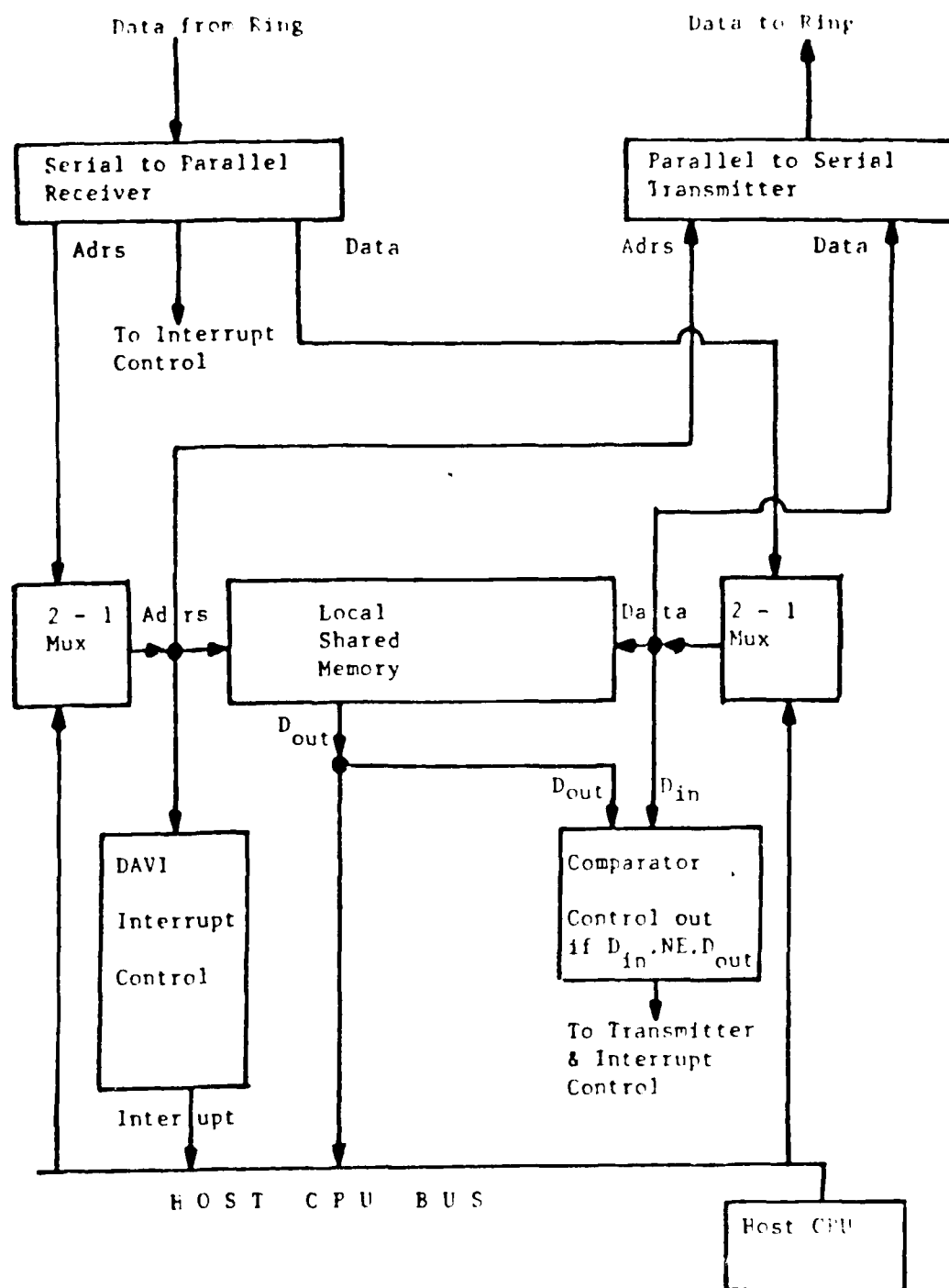


Figure 2.2 Local Shared Memory Serial Interface Unit Block Diagram

DAVI unit is labelled the Local Shared Memory Serial Interface Unit (LSMSIU).

SERIAL COMMUNICATION SCHEME

The serial format is basically an address followed by data. For a 1 Mbyte shared memory system, there would be 20 address bits followed by 8 data bits. Each node on the network must be identified by a number and this number must be appended to each message the node generates.

SERIAL/PARALLEL TRANSMISSION AND RECEPTION

All address and data inside the node and the host computer is operated on in parallel format. All network transmissions are formatted serially. Therefore, a serial to parallel converting receiver and a parallel to serial converting transmitter must interface the network and node hardware. The transmitter must append its node number to node generated messages. When receiving serial information, the node number field must be examined to see if the message should be forwarded.

DISTRIBUTED SHARED MEMORY

The shared memory emulation is implemented as a distributed shared memory. An identical configured copy of the shared data variables shall reside at each networked computer node in memory labelled local shared memory.

INFORMATION DETECTION

Information is generated only when a host computer is writing new data to a shared memory location. Information is detected by a

difference in any bit between the data currently stored in a shared memory location and the data the computer is currently writing to that location. Once detected, the new data together with its address is loaded into a parallel/serial transmitter buffer.

DATA ADDRESS VECTORED INTERRUPT SCHEME

When new data is received from the network, the data address is employed to fetch an interrupt vector from the interrupt vector lookup table. An interrupt is then requested and the vector guides the computer to the appropriate code to process the new data.

The basic concept structure of the entire system should now be evident.

3. NETWORK HARDWARE CONCEPT ELABORATION

The following hardware concept discussion describes a complete generic configuration of a data-driven information network components for real-time multiple computer systems. Figures 2.1 and 2.2 illustrated a block diagram of an entire Data-Driven Serially-Linked Shared-Memory ring network.

SERIAL COMMUNICATION SCHEME

All serial transmissions are unidirectional point-to-point at an approximate speed of 10 to 100 Mbits/second. The transmission speed is selectable so as to obtain a desired system message capacity. The format of the transmitted information is shown in figure 3.1. Basically the information address followed by the information data is transmitted. A source number tag, parity bits, and message synchronization bits are concatenated to the information address and data. The source number tag indicates which LSMSIU is the author of the transmitted information. Of course, a network of only two LSMSIUs does not require a source number tag.

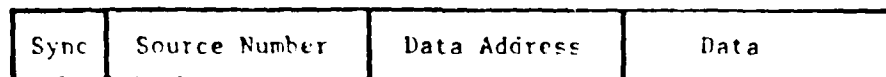


Figure 3.1 Serial Information Word Format

Fiber-optics are employed herein as no more than a very high speed long distance serial transmission medium. However the ring architecture was partially chosen due to the ease of implementation with fiber-optics. Some applications of this architecture may realize that very high speeds are not required and conventional electrical mediums may be sufficient. A parallel transmission scheme particularly for short distances may be employed with this data-driven architecture.

The system designer or system implementer must decide transmission characteristics given the physical configuration of the system to be designed. The system transmission parameters are: (1) Physical distances between computer network nodes, (2) Number of network nodes, (3) Speed of light in a fiber being approximately two-thirds of light speed in a vacuum, (4) Message duration time which is the product of transmission speed and the number of bits in a message, and (5) Number of messages per second transmitted by each node.

Figure 3.2 illustrates the ring physical layout for the message travel time analysis graph presented in figure 3.3. This graph illustrates the significance of message transmission time as function of the fiber distance. The system designer must compute the required transmission rate (Mbit/sec) from the particular fixed variables of distance between computers and system message capacity. Considering the actual transmission time, it should be noted that the transmission rate becomes less significant as the intercomputer distance increases. And it should also be noted that higher transmission rate equipment is more expensive and has a higher bit error rate than slower fiber-optic systems. The other design consideration is the average and burst rate that a host computer can write data, with and/or without information

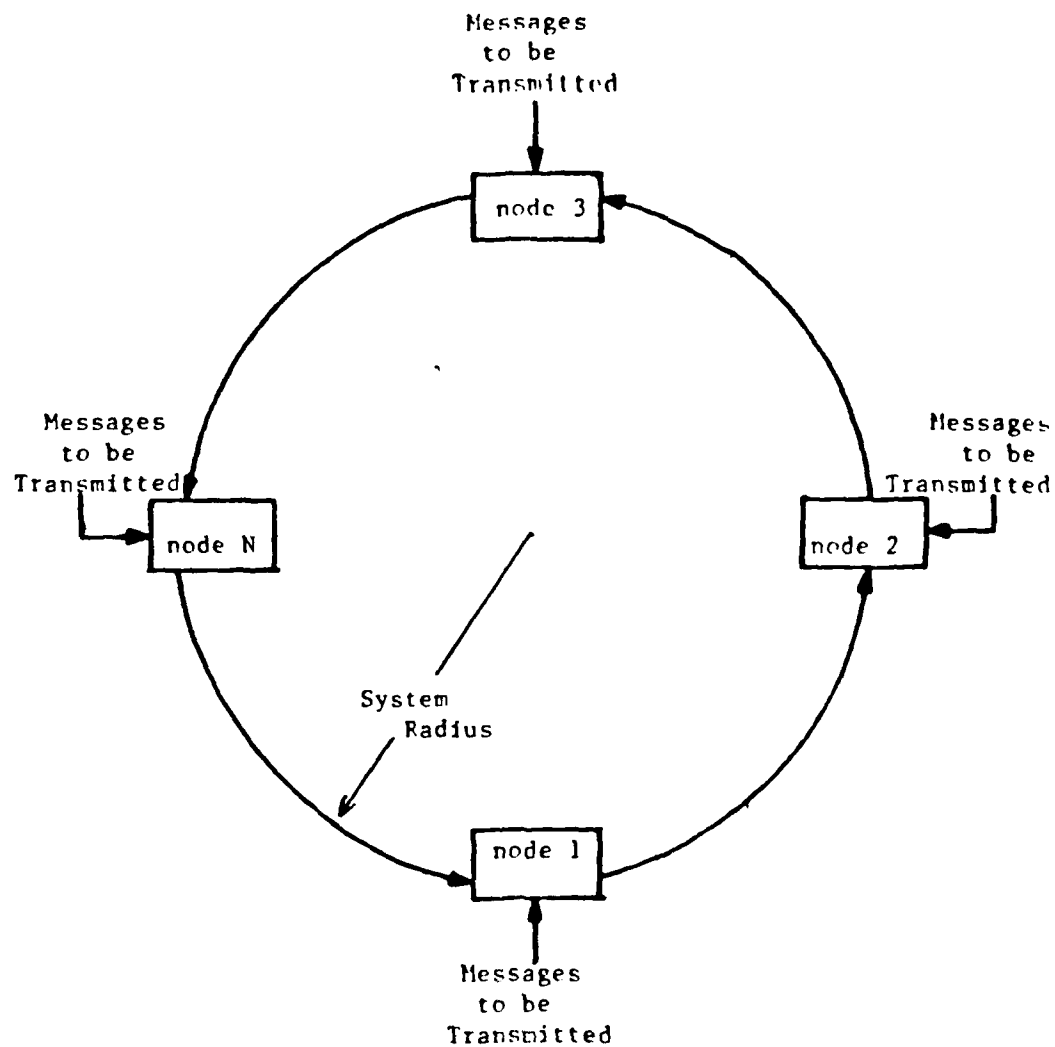


Figure 3.2 Ring Network Physical Layout

curve num	num of nodes	mes xmit rate (khz)	mes length (microsec)
A	5	0.1	0.5
B	5	0.1	10.0
C	5	10.0	10.0
D	5	10.0	50.0
E	50	0.1	0.5
F	50	0.1	10.0
G	50	10.0	10.0
H	50	10.0	50.0

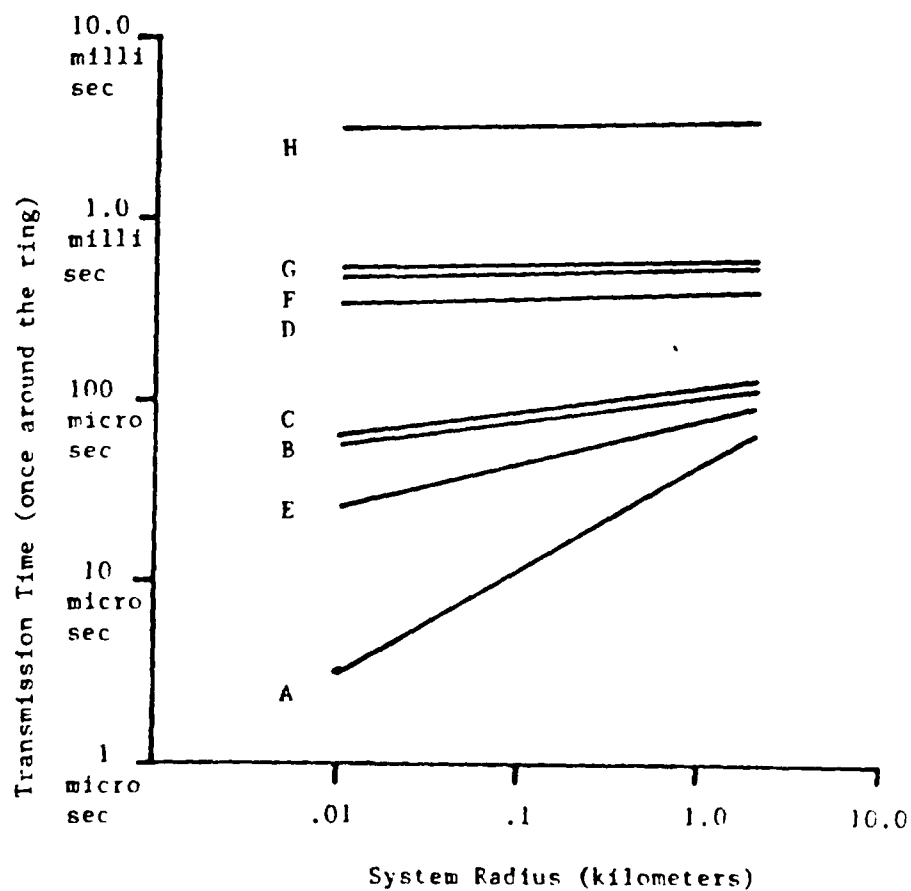


Figure 3.3 Serial Message Transmission Time

content, to the LSMSIU. This consideration takes into account, besides the serial transmission parameters, the local shared memory speed, information detection and subsequent control time, incoming data rates from the network and the actual rate of information creation due to the dynamics of the real-time software.

LOCAL SHARED MEMORY SERIAL INTERFACE UNIT

The Local Shared Memory and Serial Interface Unit residing on the processor bus of each networked computer is the respective computer's interface to the network. A block diagram of the LSMSIU is contained in figure 2.2.

Each computer sends information to the network through its LSMSIU. The LSMSIU contains a local copy of shared data and appears to be identical to the host CPU as would be a conventional shared memory system. When the host CPU reads shared data, he does so by reading his local copy of shared data from the LSMSIU. Shared data reads assert no external serial network action whatsoever. When the host CPU writes to shared data, the LSMSIU checks for possible information content of the written data. If information content is detected, the information is transmitted for reception by all LSMSIU's in the network. Information is present when the written data is different than the data previously residing at the concerned memory location.

The LSMSIU also receives information from other LSMSIU network nodes. When the information is received, the Local Shared Memory (LSM) is updated and a DAVI interrupt is generated. The vector of the interrupt directs the host CPU program counter to the software program that must (or may) respond to the incoming information. This construct permits the data-driven (or information driven) aspects of this

project.

The local shared memory interfaces to the host CPU bus by conventional CPU bus to memory communications. Essentially this is a parallel interface with address, data and control lines. This circuitry of the LSMSIU interface is nothing unconventional. The LSM word size is the same as the host CPU and the word count must be large enough to accommodate the requirements of the software task at hand. The LSM has four access ports; one each for the host CPU, the serial transmitter, the serial receiver and the information detection port. The information and transmitter ports are read only with the receiver port being write only and the host CPU port being read/write.

Information is exclusively generated when the host CPU writes new data to a memory location; especially in this case to an LSM address location. It is possible for the host CPU to write data to a memory location that is not new. That is, the written data is the same as the data residing in the particular memory location before the write occurs. In this case, information is not generated. This situation often occurs with real-time system computations. By design, the LSMSIU shall transmit information on the external information bus as a result of the host CPU writing information to the LSM.

To detect for information occurrence a data comparator is employed. When the host processor performs a write to local shared memory, the following sequence occurs:

- (1) The write action is detected.
- (2) The LSM reads and then stores separately the appropriate memory location contents for the impending compare action.

- (3) The write data is then compared to the existing data from step 2.
- (4) If the data is new, the data is written to the LSM and is sent along with its address to the serial transmitter where it is transmitted on the information bus.
- (5) If the data is not new, the sequence is terminated.

The serial transmitter is essentially a shift register and a fiber-optic transmitter which transmits the LSMSIU generated information to the downstream node. The transmitter receives the information from the LSM. The transmitter can also contain a First-In-First-Out (FIFO) buffer if the information is generated faster than the network transmission speed.

The Data Address Vectored Interrupt mechanism involves the fiber-optic serial receiver, the host processor interrupt circuitry coupled with the vector address lookup table. The receiver is a shift register that also contains a FIFO buffer to accommodate high burst data rates. When information is received, the LSM is updated and a host CPU interrupt is generated. The vector of the interrupt is determined by the contents of the vector address lookup table. The address portion of the information received is employed for writing into the LSM as well as retrieving the vector address from the table.

The table is loaded prior to run time with vectors for the DAVI interrupt. The contents of the table directly point to the address of the software program that must respond appropriately to the incoming information. The DAVI circuitry may be programmed to respond to the echoed reception of its own generated information.

4. DATA-DRIVEN SOFTWARE CONCEPT ELABORATION

Software for this scheme is completely interrupt driven. The interrupt occurs with the receipt of each word of information. It is not hard to reason that the subsequent scheduling of software model responds to data-driven interrupts can be complicated. The success of the scheme will depend on many factors including: (1) Interrupt context switching time, (2) Ability to structure a large program as a many module program with execution time efficient modules, (3) Ability to separate time dependent and independent modules, and (4) Ability to determine the required dynamic workload of the real-time software in order to program time dependent sensitivities. The data-driven software concept can be explained by the following component discussions followed by a flight simulator example.

REAL-TIME OPERATING SYSTEM

An operating system needs to be designed exclusively for real-time applications of the data-driven system. It must schedule interrupt-responding program execution, set dynamic interrupt priority levels, swap memory in from secondary storage. In large systems where many interactive users may be present background tasks must also be scheduled.

SECONDARY MEMORY UTILIZATION

With conventional synchronous real-time systems the input to output response time averages one-half the frame time or about 10 to 15

milliseconds. These systems can not use secondary swapping memory since there is not enough time to detect that an input is requiring a non-resident program to be executed, swap in the program and then execute the program, all in 10 to 15 milliseconds. If the response is required for human consumption, a 15 or 100 millisecond response time is acceptable; the human can not tell the difference. With the response time of concern, there is enough time to swap in programs for subsequent execution if the swap is commanded at the time of input information creation and not at a time later when the input is polled. The swappable programs should be the infrequently used or slower required response time programs. The memory swapping techniques are a typical concern of operating system technology.

DATA FLOW GRAPHICAL PROGRAMMING

To illustrate the interdependencies of the shared data and corresponding data-driven software modules, a graphical approach is needed in the programming-edit phase of the software development. This graphical approach is similar to the conventional Data Flow graphical approach to programming as illustrated in Soh [2].

HIGH ORDER LANGUAGE IMPLEMENTATION

Suppose that we are programming the proposed data-driven system in Fortran or a similar language. In addition to the real-time operating system, the program would be a collection of DAVI interrupt handlers. However, additional language constructs are needed to simplify the programming process. A software subroutine would handle each DAVI. Instead of a 'return' statement, a 'return from interrupt' statement would be needed at the end of each interrupt handling subroutine.

DATA-DRIVEN REAL-TIME FLIGHT SIMULATOR SOFTWARE

Consider the real-time flight simulator of Figure 1.6. The corresponding software flow graph is illustrated in Figure 4.1. Whenever a cockpit input variable changes, the respective software module is executed. Subsequent modules along the flow graph are executed until a module does not produce a change in its output. The flow then dies at this point. If the position, attitude or roll outputs change, the out-the-window scene is commanded to be redrawn.

The flow graph for the simulator appears similar to an analog computer diagram including feedback constructs. This implies that along the flow, summations and integrations are performed. This similarity is not unintentional. Allowing for analog and digital signal resolution differences the flow graph technique along with data-driven constructs appears to be a viable approach to construct a real-time near-continuous system from digital components.

Now, what happens if none of the cockpit inputs change? Surely the process must not freeze. The answer ... there is always one input variable that never stops changing. That variable is the Time . The quantizing of this variable for the purpose of data-driving time dependent modules is the task of the Simulation Dynamic Workload Assessment (SDWA) module. The SDWA module is tightly coupled to the real-time operating system. This module must determine the sensitivity to time for each of the time dependent modules in the system. The sensitivity may be dynamic. That is to say that the OTW scene must be updated more frequently when flying low and fast than when flying high and slow. Admittedly, this time dependency calculation and module scheduling process appears to be a step backward toward the synchronous

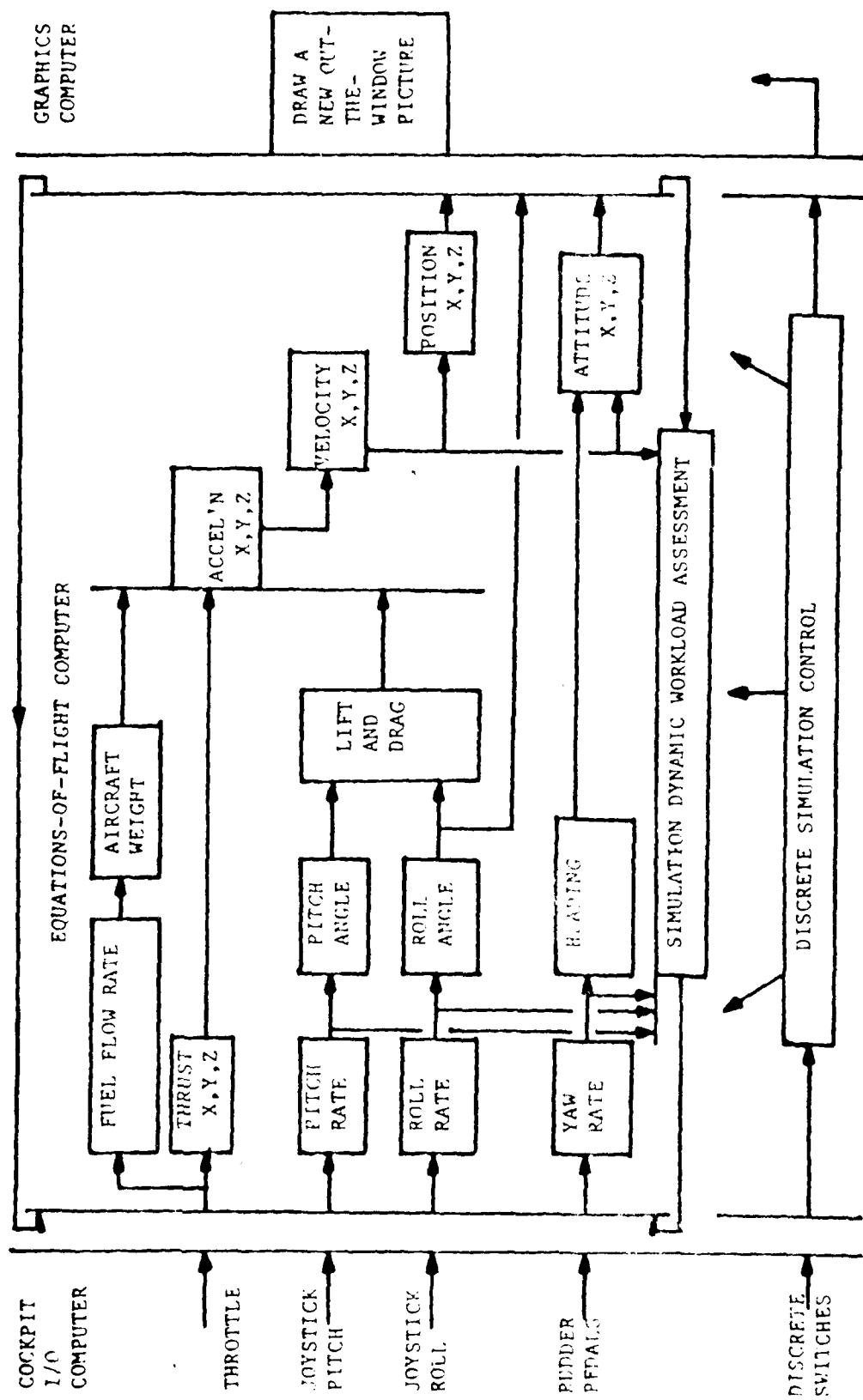


Figure 4.1 Flight Simulator Data Flow Graph

repeating scheme we are trying to avoid. However, time is treated only as a dynamic shared variable data-driving the software. The workload term is a result of fidelity maintenance of the simulation. To maintain a reasonable fidelity with a desire increase the system efficiency under a data-driven scheme, the software must 'work' harder as the pilot flies lower and faster.

5. ADVANCED SYSTEM ELABORATION

During the course of the research performed for this project it became apparent that the project could be eternally enhanced. The basic structure is firm but capability extensions and applications appear unbounded. A discussion of several of the extended capability applications follows.

SINGLE COMPUTER SYSTEM

It is possible that this data-driven system can be implemented within a single computer system. The serial network transmission scheme is not required. The LSMSIU circuit would receive its own transmissions as if the data were coming from another computer. A diagram of the hardware is shown in figure 5.1.

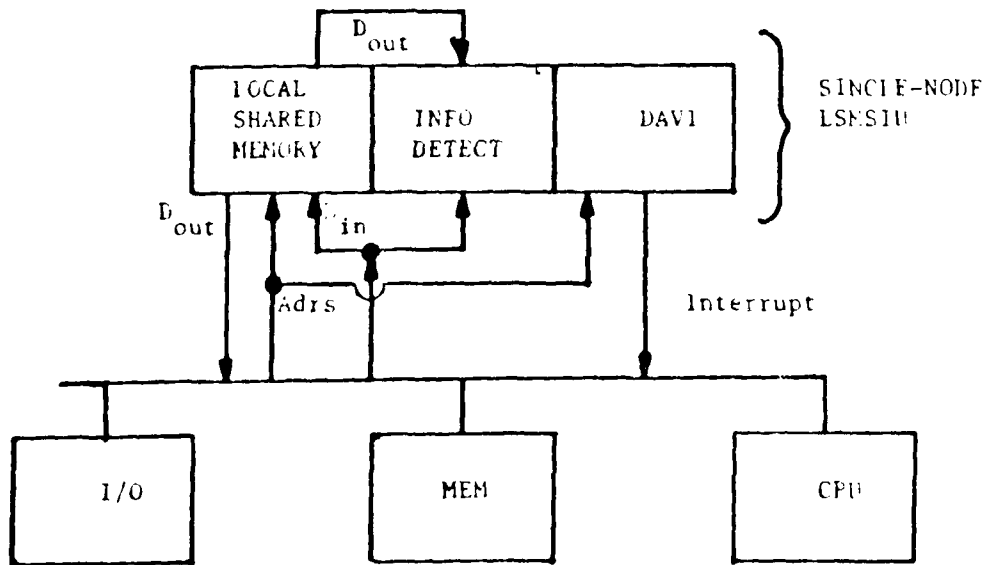


Figure 5.1 Single-Node Data-Driven Computer System

Data address vectored interrupts responding to changing data would occur in identical fashion to the multiple computer data-driven system. The real-time operating system will be more complicated and must be able to handle interrupt driven recursion resulting from software modules with tight feedback. These modules have an input which is directly driven from the module output.

DATA VARIABLE SENSITIVITY

The system user will most likely not desire an interrupt to occur when each variable changes by a small amount. For example, an interrupt is not needed when the aircraft simulator altitude changes from 30,000 to 30,001 feet. Likewise, the OTW scene need not be updated for this minute altitude change. Therefore, a data variable sensitivity parameter needs to be introduced.

Implementation of the sensitivity would require 2 additional local memories; one to store the sensitivity parameter for each data variable and one to store the value of the data variable that last instigated a DAVI. The original local shared memory would still contain the latest and most precise value of each data variable. It is noted that the sensitivity parameter can be implemented as an absolute value or as a per cent value change in the dynamic data variable, e.g. a DAVI may occur when the aircraft altitude changes by 300 feet or 1 percent. The sensitivity parameter should be adaptive to dynamic conditions during run time. For example, as the altitude decreases, the sensitivity should most likely increase. The LSMSIU with the sensitivity memories is shown in figure 5.2.

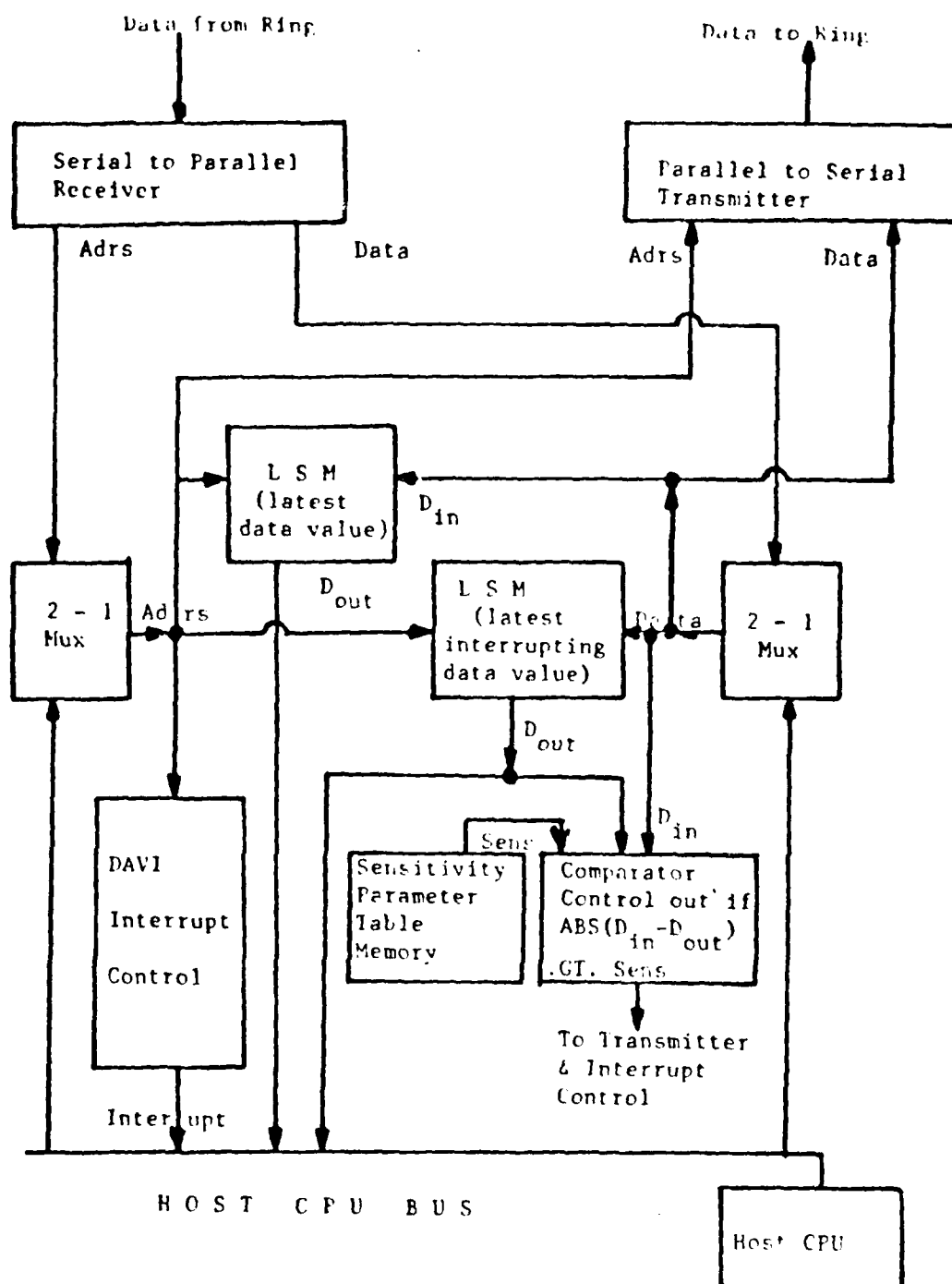


Figure 5.2 LSMSIU with Data Sensitivity Parameter Implementation

REAL-TIME GRAPHICS IMPLEMENTATION

In many of the video (raster or stroke) graphics units, out the window scenes are generated by a series of line figures. The lines are periodically redrawn to give the effect of smooth motion. However, if the motion is slow, identical picture are unnecessarily updated and redrawn. And if the motion is very fast, the lines jump in quantum leaps due to slow update rates.

Data-driven concepts can solve this problem by introducing a data-driven scene dynamic factor. This factor would indicate the absolute angular change in position of the physically nearest item in the viewport. Then as the angular position changes by an amount greater than the angular sensitivity of the eye or resolution of the video display, a DAVI would occur forcing a scene update. Also to be considered is the speed of the graphics unit. There is no need to command picture updates faster than the unit can draw a single picture.

MULTI-VARIABLE SYNCHRONIZATION

A special situation arises when one software module has several inputs that are derived from a group of data-driven outputs from another module such as the velocity calculation in figure 4.1. It is required that the module is scheduled only once when any or all of its inputs change. It is undesirable for the module to be executed unnecessarily.

Therefore, for modules with several data-driven outputs, an output 'flush' command is required. All outputs with changed values will be transmitted contiguously with the flush command. If no outputs have changed, nothing will happen with the flush command. In this mode, a data receiving module will only be executed once for the output group.

DATA TAGGING

Data tagging is a technique that adds to each data variable a field indicating the type and precision of the variable. Examples are: a 3 byte integer, a 8 byte floating point number, a single byte character or a single byte logical. This technique saves space in the instruction codes of the computer. For example, the op-code indicates add this to that, when fetching the operands, the tag field indicates whether this and that are integers or reals, then according to the tag field the proper integer or floating point mathematics is performed. Normal shared memory alone does not know the type of the variable being stored. Data tagging could assist the information detection, transmission and reception process. To detect the degree of change in a variable, the information detection unit must know the starting address of the variable as well as the type of the variable. This is especially true for variables that are longer than one computer word.

SERIAL TRANSMISSION ERROR DETECTION AND CORRECTION

With this system design, network transmission errors are disastrous. In synchronous simulation systems where all variables are update continuously, a single variable error would vanish after one frame time. But in this system a single error may be permanent. Therefore effective error detection and correction schemes are required.

With the ring network architecture, messages are echoed back to the originator. The originator can check the echoed message for errors. A retransmission scheme is required to correct the originator detected errors. When the receiver detects an error and does not detect a retransmission, he may request a retransmission via a

data-driven retransmission variable. Implementation of the retransmission detection circuit appears cumbersome but possible.

Considering that 100 Mbit/sec fiber-optic links transmit 100 bits in a microsecond, there is time available to include many error detection and correction code bits.

6. PROTOTYPE DESIGN

A two node system was designed to support the validity of the major architecture design. The system was designed using readily available Motorola MC68000 microprocessor computer boards which are labelled 'Educational Circuit Boards (ECB).'

Each ECB contains: (1) a 4 Mhz MC68000 CPU, (2) 16 Kbyte Read-Only-Memory (ROM) containing the supplied operating system, (3) 32 Kbyte Random Access Memory (RAM), (4) 2, RS-232 serial ports, one for the terminal, one for the host computer if required, (5) parallel printer interface with timer. Each Local Shared Memory Serial Interface Unit hardware designed for this project includes (1) 256 byte RAM, (2) 1 Mbit/sec electrical serial interface, (3) information detection circuitry, and (4) microprogrammed control unit. A block diagram of the MC68000 ECB and Network Node is shown in figure 6.1. Shown on the left of the buffer is the MC68000 ECB and on the right is the LSMSIU. The LSMSIU has no DAVI vector lookup table. This table is to be implemented in the ECB memory. When serial information is received, the ECB is interrupted and ECB hosted software reads the address of the information data. This address is then to be used to vector through the ECB based DAVI lookup table.

The documentation illustrating all features of the M68000 ECB is available in [7]. The hardware schematics for a single LSMSIU ring network node is contained in Appendix I. The microprogram operations are listed in Appendix II.

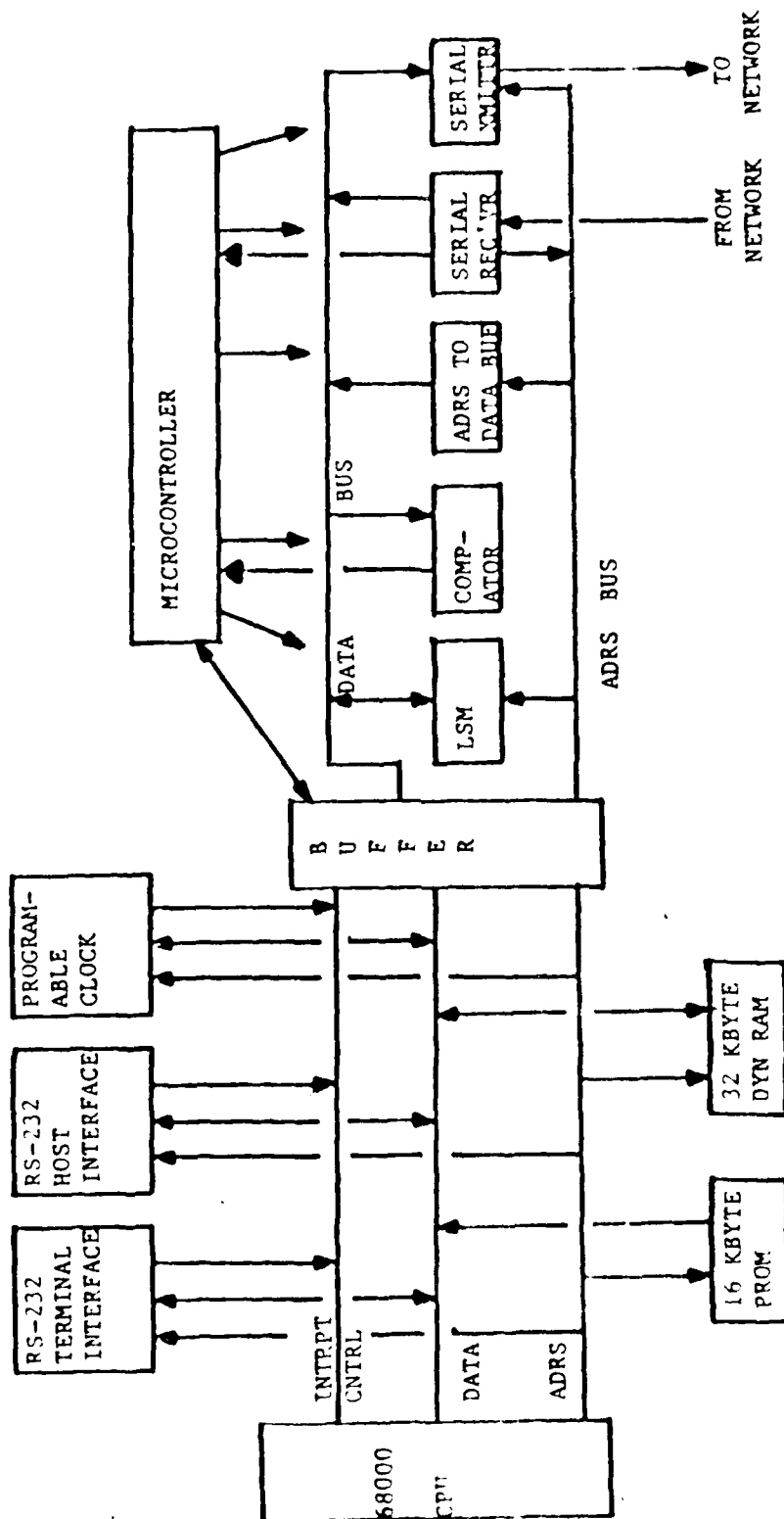


Figure 6.1 M68000 Based Data-Driven Network Node Prototype Design Block Diagram

HARDWARE STRUCTURE

The following discussion is made referencing the Prototype Hardware Schematics in Appendix I. The Block Diagram illustrates the entire interconnection including the:

- (1) Data Bus and Address Bus Structure including the Local Shared Memory, Comparator and Adrs to Data Buffer,
- (2) Microcontroller,
- (3) 68000 ECB to LSMSIU Interface Buffer,
- (4) Parallel to Serial Transmitter, and
- (5) Serial to Parallel Receiver,

The LSMSIU data bus is an 8 bit tristate bus. The address bus is also an 8 bit tristate bus. The LSM is implemented by 2, 2114 1k by 4 bit RAM's. For easy implementation with an 8 bit address, only 256 of the 1024 bytes are used. The comparator is implemented by 2, 7485, 4 bit comparators. A latch to the B front end of the comparator is employed while the data bus holds the A front end data when making comparisons from the ECB write data to the 2114 memory contents. The A=B comparator output anded with the microcontroller compare enable signal are employed to direct the LSM write sequence to the new data or no new data write subsequences. The 74ls374 address to data buffer is employed when reading the address of the serially received information.

The microcontroller is implemented by 2 2716 EPROM's giving a 2k by 16 bit controller. The outputs and sequences are described in Appendix II

The 68000 ECB to LSMSIU Interface Buffer buffers address, data and read, write and interrupt control signals between the ECB and LSMSIU. The El* line is a read/write strobe line asserted when the ECB is

communicating with the LSMSIU. The DTACK and DTACK PIT* signal are slave responses when the read or write process is complete. The LSM IRQ* or 6800 IRQ* signals are asserted when the LSMSIU interrupts the ECB. The A09 address lines indicate to the LSMSIU whether the ECB is communicating to the LSM or the vector address 74ls374 buffer. A09 = 0 indicates the LSM should be activated while a 1 indicates the vector address buffer assertion.

The Parallel to Serial Transmitter consists of 2, 74ls299 8 bit shift registers and 1 74ls3 4-1 multiplexer. The serial out format is in a simple binary format. It starts with a 1 microsecond high level sync pulse followed by 1 microsecond low level followed by 8 bits of address, MSB first and 8 bits of data, MSB first. When the transmission is complete the bus returns to a low level. The shift registers and multiplexer are completely controlled by the microcontroller.

The Serial to Parallel Receiver performs the complement process to the transmitter. The receiver operates at a 16X sampling frequency and employs 74ls63 counters frequency division to shift in the address and data. The 2 shift registers clock in the address and data. When the serial receive process is complete the receiver requests an ECB interrupt and the receiver remains busy until the serial data address vector is read by the ECB.

HARDWARE OPERATIONS

The following operations can occur within the LSMSIU:

- (1) ECB read of the Local Shared Memory,
- (2) ECB write to the Local Shared Memory with:

- (a) a data change with subsequent serial information transmission or,
 - (b) no data change,
- (3) Serial reception of information including a data write to the Local Shared Memory with an ECB interrupt, and address vector read.

When the ECB commands a read of the LSM, the microcontroller starts in the LSM read sequence. The 8 bit address lines are asserted directly from the interface buffer. The data line buffer is asserted with the proper data direction. The 2114 RAM's are enabled driving their contents on the data lines. DTACK is then asserted ending the sequence.

When the ECB commands a write to the LSM, the microcontroller starts in the LSM write sequence. The LSM data using the ECB driven address lines is loaded into the comparator B latch. Then the ECB driven data is asserted on the data bus and the comparator compares the two data. If the data are identical the write old data subsequence is commenced which is implemented via an immediate DTACK. If the data are not equal, the new data is loaded into the LSM and the serial transmitter and a lengthy serial transmit sequence is begun.

The Serial to Parallel Receiver shifts in address and data information without microcontroller control. When the receive process is complete the serial receive done microcontrol sequence is begun simultaneously with interrupting the ECB. Via the microcontroller the received information is loaded into the LSM and the address information is loaded into the address to data buffer for the vector read process. The vector read process is asserted similar to the LSM read process and differs in that address bit 09 is asserted which commands the

information address to data buffer to transmit its data at this time.

7. SUMMARY AND CONCLUSIONS

An integrated hardware and software architecture encompassing a data-driven information network for real-time multiple computer systems has been introduced. Innovative highlights of the architecture are:

(1) A shared memory interface scheme for local area (less than 10 km) networked computers involving fiber-optics.

(2) Dynamic data-driven scheduled execution of real-time software.

(3) A scheme which enables swapping of programs in a real-time environment.

The concepts of the architecture have been illustrated in a 'cookbook' fashion with a flight simulator system example as an educational aid. The actual efficiency and community acceptance of the scheme remains to be determined. The utility of the hardware scheme with or without the data-driven software constructs appears beneficial for real-time systems such as multi-computer flight simulators. The scheme as it has been disclosed could be employed in non-real-time applications of multiple computer systems. However, due to inefficiencies of the scheme when sending a 'block' of data through the shared memory, this system will most likely not be used in non-real-time environments. Although, with some modifications slightly impacting real-time network performance, the scheme could be made useful when sending blocks of data.

To the application programmer the new hardware system is a simple

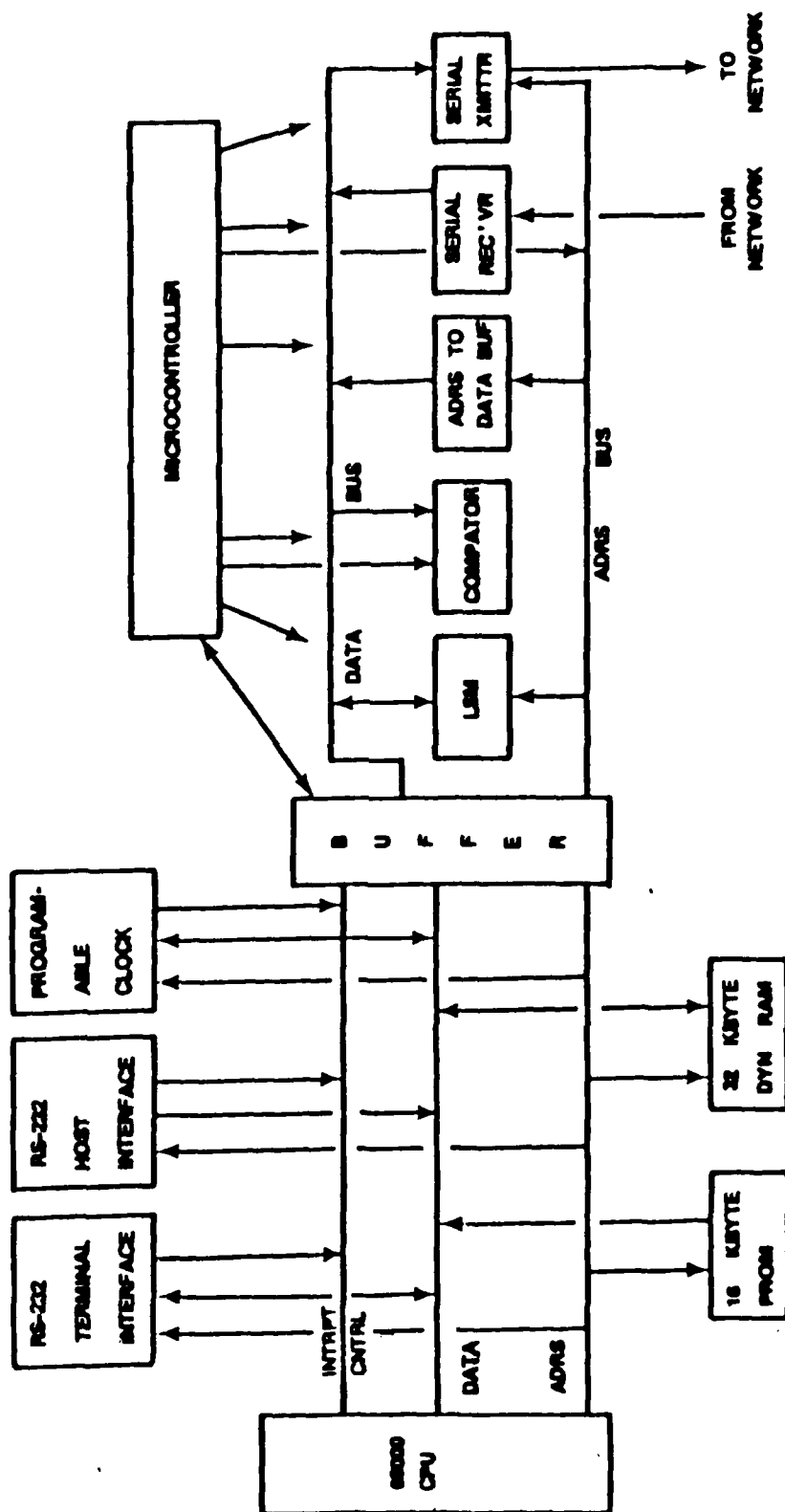
shared memory for data communication. The programmer must implement shared data programs with proper synchronization. Multiple computers simultaneously writing to the same location is not allowed. In this event, different copies of a shared memory data item may have different values which would be catastrophic.

The software scheme is immature at this point. Extensive research beyond the scope of a thesis project is required. The complexity and utility factors of an implemented data-driven real-time operating system remain to be evaluated. These factors need to be weighed against those in current synchronous real-time operating systems. The main idea behind the data-driven real-time system is to execute software modules only when a data input to the module changes value. This simply increases the efficiency of the system. The scheme can become very complex with interactive modules containing multiple data inputs and outputs. Execution control of these modules must also consider outputs from other modules queued for running on this or other networked computers, as well as maximum and minimum rates of execution (i.e. module I/O sensitivity to time). Without careful operating system design, modules with multiple inputs arriving from several external computers may execute much more often than necessary.

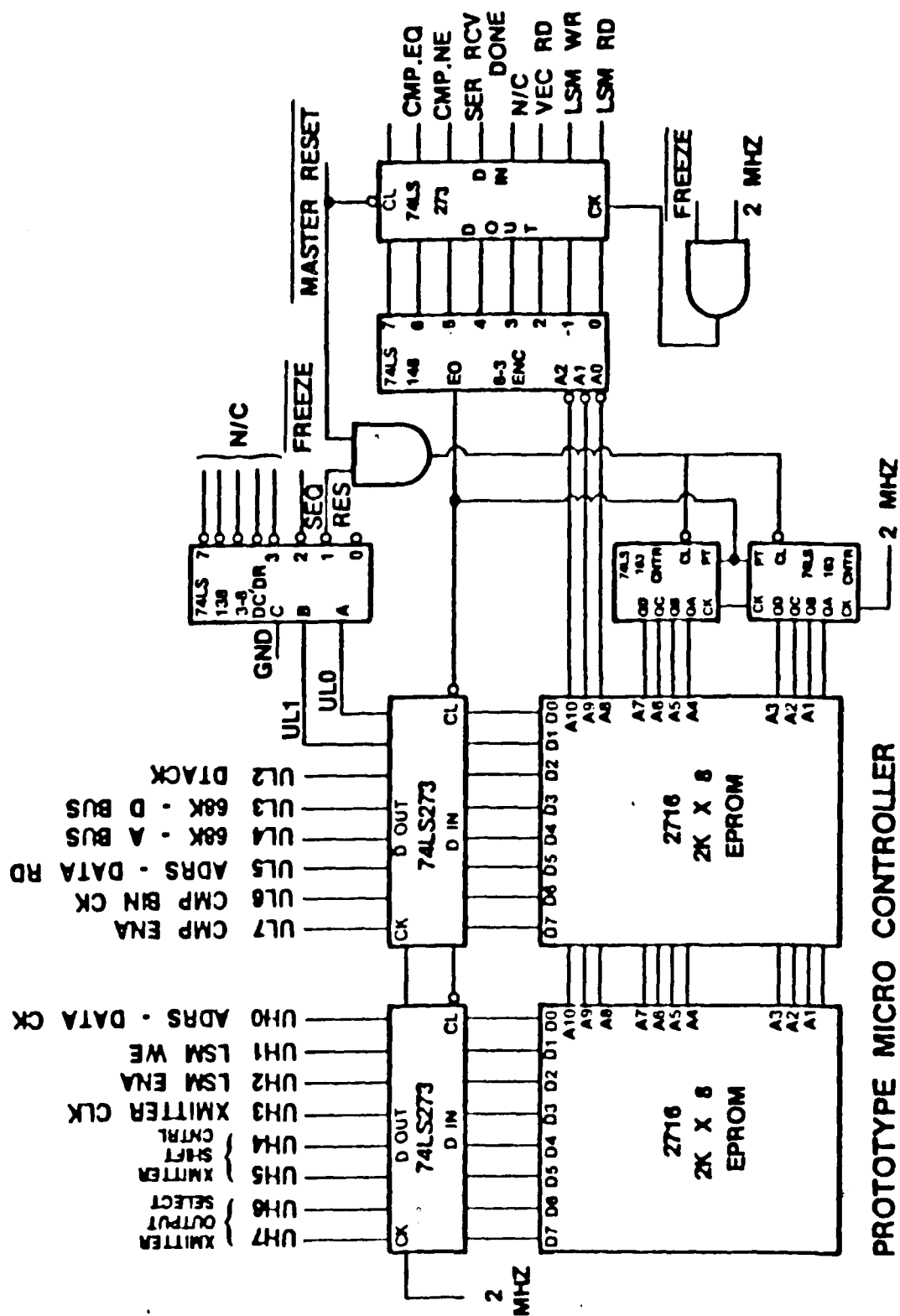
The utility of the proposed data-driven replicated shared memory network with fiber-optic serial transmission appears very high. No hardware or software problems are expected with implementations of the network and conventional synchronous real-time software. As indicated, real-time data-driven operating system with the proposed network requires further research.

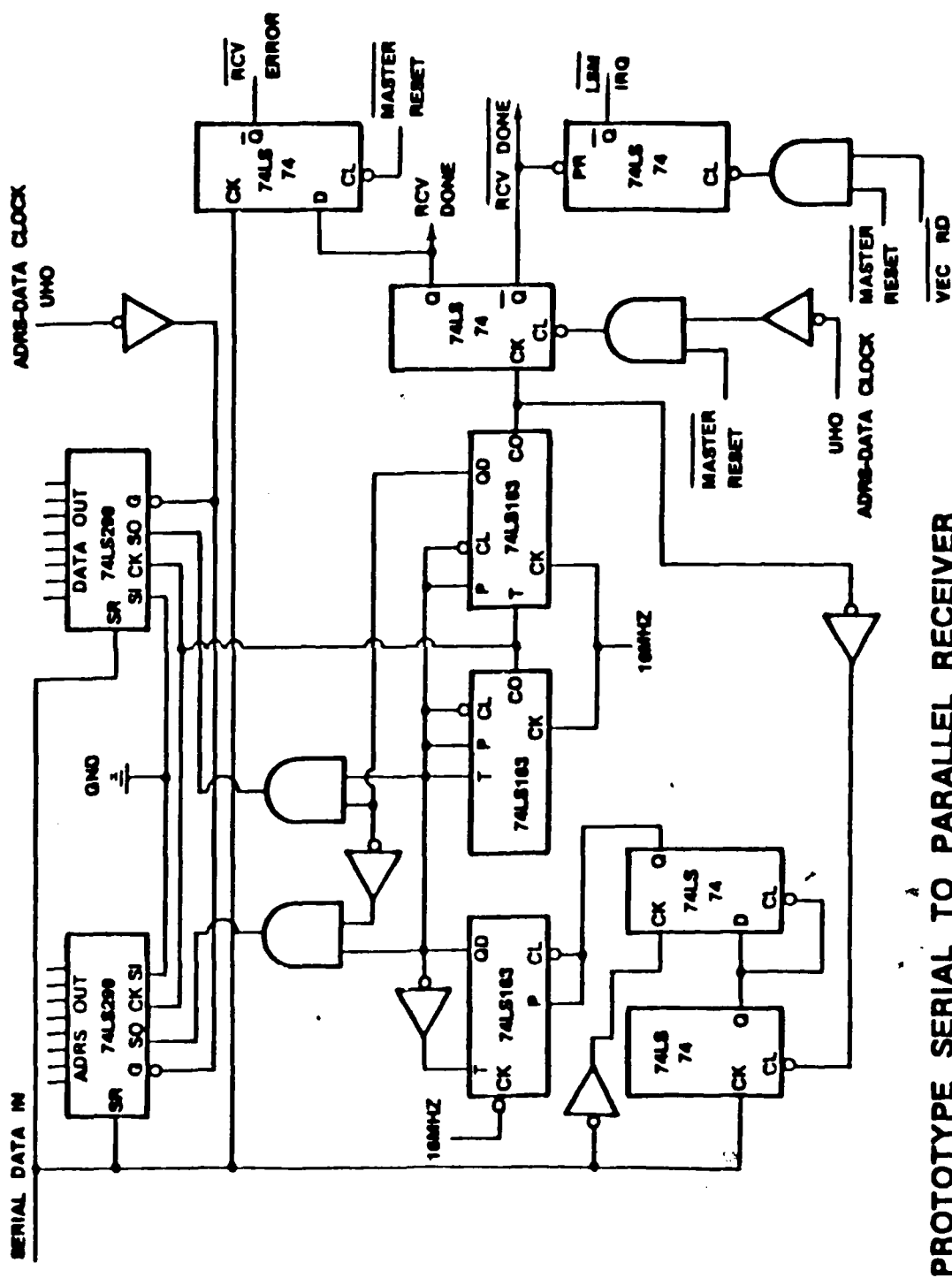
APPENDIX I
PROTOTYPE HARDWARE SCHEMATICS

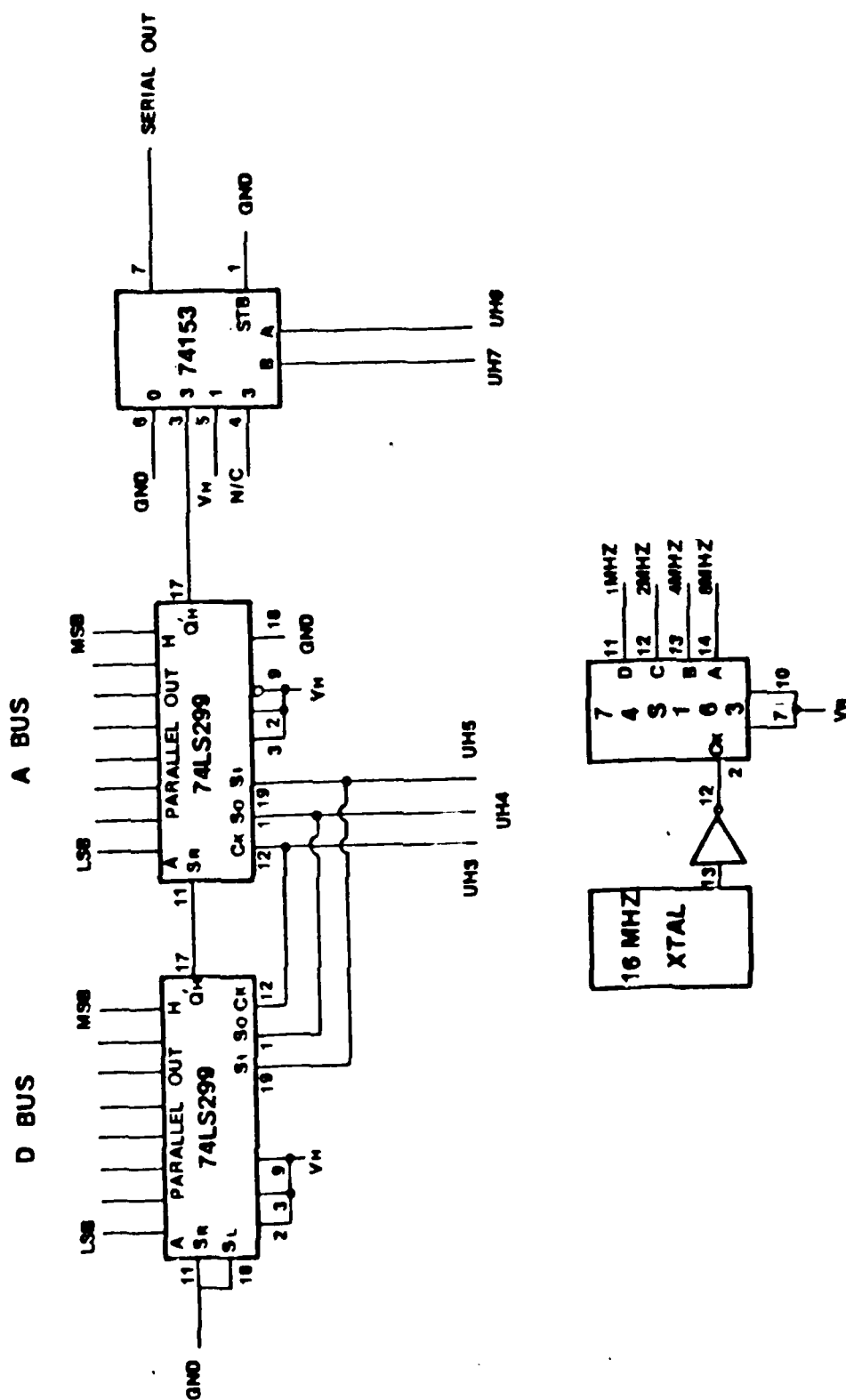
13



**M68000 BASED DATA-DRIVEN NETWORK
NODE PROTOTYPE DESIGN BLOCK DIAGRAM**







PROTOTYPE PARALLEL TO SERIAL TRANSMITTER AND CLOCK DIVIDER

APPENDIX II
PROTOTYPE MICROCONTROLLER FIRMWARE

The microcode sequences illustrated on following pages control the hardware sequences labelled:

- (1) Writing data to the local shared memory main sequence,
Base address 600.
- (2) Writing new data to the local shared memory subsequence,
Base address 200.
- (3) Writing old data to the local shared memory subsequence,
Base address 100.
- (4) Reading local shared memory main sequence,
Base address 700.
- (5) Reading the new data vector address,
Base address 500.
- (6) Serial Receiver done handler,
Base address 300.

The microcontroller is implemented with 2, 2K by 8 bit EPROMS and there are 16 output control lines divided into high and low 8 bits. The lines are defined as follows:

High 8 bits:

Bits 6 and 7 control the A and B select inputs to the output of the serial transmitter. There are 3 output possibilities defined which are:

Bit 7, Bit 6, Output definition

0	0	low (or 0)
0	1	high (or 1)
1	0	undefined
1	1	shift register output is serial transmitter output

Bits 4 and 5 control the transmitter shift register state which are;

Bit 5, Bit 4, Shift Register State

0	0	Hold
0	1	Shift Right
1	0	Shift Left (not to be used)
1	1	Load

Bit 3 is the clock input to the shift register.

Bit 2 is the local shared memory RAM enable line.

Bit 1 is the local shared memory RAM write enable line.

Bit 0 enable the 74ls374 buffer to latch on to the address of the serially received data.

Low 8 bits

Bit 7 enable the information detection data comparator.

Bit 6 clocks in the data to the B input of the comparator.

Bit 5 enables the 74ls374 buffer to transmit the serially received data address on the data bus lines for host computer read.

Bit 4 enables the host computer address to be transmitted on the address bus.

Bit 3 enables the host computer data to be transmitted or received to/from the data bus.

Bit 2 enables the data transfer acknowledge (DTACK) signal to the host computer signifying the end of a data transfer to/from the host.

Bit 1 and 0 control the input to the sequencer which is defined as follows:

Bit 1, Bit 0, Microcontroller control

0	0	no action
0	1	Reset the sequence; go to the idle state
1	0	Freeze control inputs inhibiting a major state change
1	1	no action

MICRO CONTROLLER SEQUENCE: write old data subsequence
 BASE ADDRESS: 100 hexadecimal

HEX ADDRESS	CONTROLLER HIGH BITS 7 6 5 4 3 2 1 0	CONTROLLER LOW BITS 7 6 5 4 3 2 1 0	CONTROLLER ACTION COMMENT
100		1 1	perform DTACK ending seq.
01			
02			
03			
04			
05			
06			
07			
08			
09			
0A			
0B			
0C			
0D			
0E			
0F			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
1A			
1B			
1C			
1D			
1E			
1F			
20			
21			
22			
23			
24			
25			
26			
27			

MICRO CONTROLLER SEQUENCE: local shared memory read
 BASE ADDRESS: 700

HEX ADDRESS	CONTROLLER HIGH BITS	CONTROLLER LOW BITS	CONTROLLER ACTION COMMENT
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	
700	1	1 1 1	Frz, 68k to A&D bus, Ram ena
01	1	1 1 1	ditto
02	1	1 1 1 1	ditto & DTACK
03	1	1 1 1 1	ditto
04	1	1 1 1 1	Reset seq, unfz, DTACK
05			
06			
07			
08			
09			
0A			
0B			
0C			
0D			
0E			
0F			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
1A			
1B			
1C			
1D			
1E			
1F			
20			
21			
22			
23			
24			
25			
26			
27			

BIBLIOGRAPHY

1. Weitzman, C., Distributed Micro/Minicomputer Systems.
Prentice-Hall 1980.
2. Soh, S. E., Data-Driven Computing Systems: A Survey.
M.S. Thesis, Department of Computer Science, Wright State
University, 1981.
3. Swan, R. J., et al. "The Implementation of the Cm* Multi-
Microprocessor," Proceedings of the National Computer Conference,
Dallas, TX, 1977.
4. Trealeaven, P. C., Brownbridge, D. R., and Hopkins, R. P.,
"Data-Driven and Demand-Driven Computer Architecture", Computing
Surveys, Vol. 14, No. 1, March 1982.
5. Lillevik, S. L., and Easterday, J. L., "A Multiprocessor With
Replicated Shared Memory", Proceedings of the National Computer
Conference, Seattle, WA, 1983.
6. Welch, H. O. and Moquin, W. A., "An Analysis of a Multicache
Shared Memory Ring Interconnection", Proceedings of the IEEE
Real-Time Systems Symposium", 1982.

7. Motorola Inc., "MC68000 Educational Computer Board User's Manual".
Copyright 1982 by Motorola Inc., Manual Number MEX68KECB/D2.